# PATHA: Performance Analysis Tool for HPC Applications

Wucherl Yoo*, Michelle Koo†, Yi Cao‡, Alex Sim*, Peter Nugent*†, Kesheng Wu*
*Lawrence Berkeley National Laboratory, Berkeley, CA, USA
†University of California at Berkeley, Berkeley, CA, USA
‡California Institute of Technology, Pasadena, CA, USA

***Abstract*—Large science projects rely on complex workflows to analyze terabytes or petabytes of data. These jobs are often running over thousands of CPU cores and simultaneously performing data accesses, data movements, and computation. It is difficult to identify bottlenecks or to debug the performance issues in these large workflows. To address these challenges, we have developed Performance Analysis Tool for HPC Applications (PATHA) using the state-of-art open source big data processing tools. Our framework can ingest system logs to extract key performance measures, and apply the most sophisticated statistical tools and data mining methods on the performance data. It utilizes an efficient data processing engine to allow users to interactively analyze a large amount of different types of logs and measurements. To illustrate the functionality of PATHA, we conduct a case study on the workflows from an astronomy project known as the Palomar Transient Factory (PTF). Our study processed 1.6 TB of system logs collected on the NERSC supercomputer Edison. Using PATHA, we were able to identify performance bottlenecks, which reside in three tasks of PTF workflow with the dependency on the density of celestial objects.**

***Index Terms*—Performance analysis, Performance evaluation, High performance computing**

## I. Introduction

Large science projects are increasingly relying on thousands of CPUs to produce and analyze petabytes of data [12][20]. These jobs usually have thousands of concurrent operations of data accesses, data movement, and computation. Understanding the performance characteristics of these complex workflows and debugging their performance issues are challenging for various reasons. The concurrent data accesses may compete with each other and with other jobs for accessing to shared data storage and networking resources. The storage and memory hierarchies on the current generation of hardware are very complex, and therefore have performance characteristics that are sometimes unexpected. Modern CPUs usually have temperature-based throttling mechanisms that could be activated to reduce the clock rate to decrease heat production, which can introduce unexpected delays. It is difficult for the application developers to anticipate all such conditions and dynamically to adjust the data processing workflows. Therefore, it is common for large parallel jobs to experience mysterious performance fluctuations. To help understand these performance fluctuations and diagnose performance bottlenecks, we have developed PATHA (Performance Analysis Tool for HPC Applications).

There have been various performance modeling works for scientific workflows under various conditions, for example, on a CPU node [24], in the Grid environment [9][11], and in the cloud environment [17][16]. However, the large scientific workflows are frequently running on a large computer with sophisticated storage and networking resources that are not easily captured by the existing models. There are a large number of other jobs competing for the same resources. For example, the computer center, NERSC Edison [1] where our tests run has about 5,000 users, and at any given time, hundreds, sometimes even thousands of parallel jobs are waiting in the batch queue to be executed. After dispatching, jobs all access the same file systems and networking hardware for I/O and communication needs. In some cases, different tasks even share the same computer node, where the computations from different jobs can affect the performance of each other. Additionally, most of the existing performance models are based on simplified models of how the underlying hardware functions. For example, the roofline model [24] is based on theoretical maximum performance of CPUs, and various I/O performance models are similarly based on the maximum performance numbers that manufacturers provide. In most cases, the observed performance is far from what could be achieved according to these simplified models. One may choose to develop more refined mathematical models, however, in this work, we choose instead to pursue empirical models based on the observed performance measurements.

The queuing system captures performance information including memory usage, CPU time, and elapsed time. However, such information is generally about the whole job, and more fine-grained information is needed to understand the individual steps of a large parallel workflow. Alternatively, the workflow management system could record the performance information of each step of a workflow [15], a profiler may be used to automatically capture detailed performance information [19], or the user may instrument selected operations with some library functions [22]. In these cases, the performance data is typically captured into log files, which require additional processing to extract the timing measurements. Once the timing measurements are available, we can apply statistical and data mining tools to study the performance characteristics.

To make the maximal use of the user's knowledge about their own applications, we enable interactive exploration of the performance data. To process a large amounts of performance

data, we use Apache Spark [27] in the backend to distribute and parallelize computational work loads. The extensive analysis capability of Spark also means that PATHA can identify performance bottlenecks through outlier detection and other data mining techniques. PATHA further invokes popular information from its visualization framework to provide interactive visualization of these bottlenecks and their dependencies. In addition, the efficient data handling capability of Spark allows PATHA to quickly integrate new performance information as it gathers from the log files.

We used the Palomar Transient Factory (PTF) [14][18] application to evaluate PATHA by analyzing performance measurement data collected on the NERSC Edison cluster. The PTF application is a wide-field automated survey that records images of variable and transient objects in the sky [14][18]. Images from these cameras are sent and stored to the NERSC Edison for processing through the near real-time image subtraction data analysis pipeline. The timestamps of the execution of each processing step in the pipeline were recorded in the database. As the performance of the PTF analysis pipeline has been optimized, its performance analysis to find hidden performance bottlenecks is particularly challenging. This difficulties is due to the manual efforts and required expertise to generate queries on the database with the requirements to avoid severe overhead on the production database shared by many users. The PTF application has been optimized to remove bottlenecks and inefficient operations by developers. Through our study, we were able to verify that the optimized versions were mostly efficient and the obvious bottlenecks were removed. Using PATHA, we identified hidden performance bottlenecks and their causes without incurring large database overhead.

The contributions are:

- Develop PATHA to handle different types of measurements from scientific applications,
- Design bottleneck detection methods in PATHA, e.g., execution time analysis and data dependency performance analysis
- Evaluate PATHA using a big data application known as PTF.

The rest of paper is organized as follows. Sec. II presents related work. Sec. III demonstrates the design and implementation of PATHA. Sec. IV presents experimental evaluations, and the conclusion and future work are in Sec. V.

## II. Related Work

Several performance tools have been proposed to improve the performance of HPC applications. Shende et al. [19] designed Tau to support monitoring parallel applications by automatically inserting instrumentation routines. Böhme et al. [5] presented an automatic mechanism which performs instrumentation during compilation in order to identify the causes of waiting periods for MPI applications. Burtscher et al. [8] designed Perfexpert to automate identifying the performance bottlenecks of HPC applications with predefined rules. Adhianto et al. [2] designed HPCToolkit to measure hardware events and to correlate the events with source code to identify performance bottlenecks of parallel applications. The detection mechanisms of these tools were heavily dependent on manually created metrics and rules. Vampir [7] uses MPI workers to parallelize performance analysis computations. However, it lacks supporting distributing the computations to multiple nodes. These performance tools lack distributing and parallelizing the computations of the analysis to large number of machines. Some tools such as Tau [19] and Vampir [7] can parallelize computational loads MPI processes, and potentially these MPI processes can be extended to distribute multiple loads. However, this extension involve significant implementation challenges due to synchronization and inter-process communication comlexities and lack of fault tolerance support. Instead, PATHA can interactively analyze the large size application and system logs of scientific workflows requiring large computation within user-tolerable latency. Furthermore, PATHA can complement these tools by providing mechanisms to distribute and parallelize the computational loads in addition to fault tolerance feature from read-only characteristics of RDDs.

There has been several performance modeling works for scientific workflows. Williams at al. [24] proposed the Roofline model about a theoretical model for analyzing upper bounds of performance with given computational bottlenecks and memory bottlenecks. Tikir et al. [23] proposed to use genetic algorithms to predict achievable bandwidth from cache hit rates for memory-bound HPC applications. Duan et al. [11] proposed to use a hybrid Bayesian-neural network to predict the execution time of scientific workflow in the Grid environment. In addition, performance models have been proposed in other domains. Cohen et al. [10] proposed to learn an ensemble of models using a tree-augmented Bayesian network on a system trace, and cluster the signatures to identify different regions of normality as well as recurrent performance problems. Ironmodel [21] employed a decision tree to build the performance model based on the queuing theory of expected behavior from end-to-end traces. These performance models are based on the simplified models or assumptions about the executions on the underlying hardwares and cluster. Our performance analysis is based on the empirical model without sacrificing the complex interactions in the executions.

Researchers have proposed mechanisms to identify performance problems in the cluster environment. Barham et al. [3] proposed to use clustering to identify anomalous requests. Xu et al. [25] proposed to find erroneous execution paths using the PCA [13] on console logs. Bod et al. [4] used logistic regression with L1 regularization on the vector of metric quantiles to fingerprint performance crisis. They used online sampling to estimate quantiles from hundreds of machines. Yoo et al. [26] adapted machine learning mechanisms to identify performance bottlenecks using fingerprints generated from micro-benchmarks. These work can help our work differentiate performance bottlenecks at cluster level and those at application level. However, they also lack support to analyze large size logs from scientific workflows.

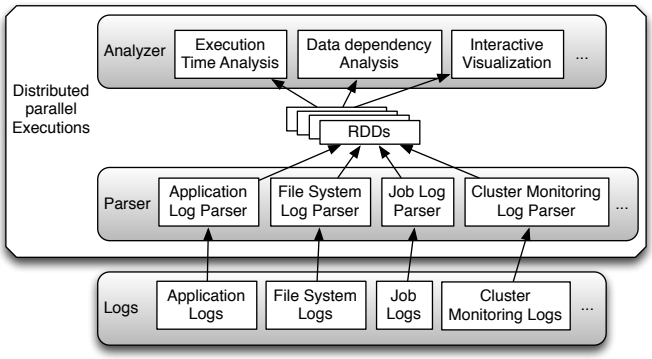



Fig. 1: The overview of PATHA.

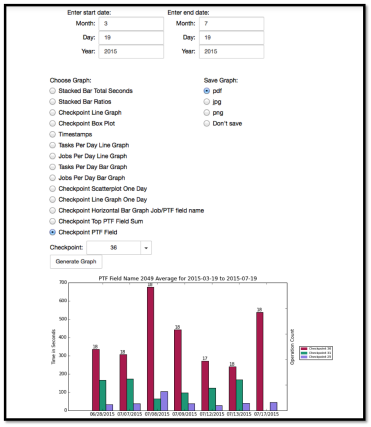


Fig. 2: The interactive visualization framework.

## III. Design and Implementation

Fig. 1 illustrates the overview of PATHA. PATHA is implemented with a fast and general framework for big-data processing, Apache Spark. It is used to distribute and parallelize computational work loads at the parser and the analyzer levels. The analyzer of PATHA supports:

- execution time analysis to find performance bottlenecks and time consuming routines in applications,
- data dependency analysis to find possible causes of performance bottlenecks
- interactive visualization synched with performance measurements

PATHA consist of parser and analyzer that are implemented with Apache Spark. At the parser level, the different types of logs stored in parallel file system or database can be loaded into distributed memory of the multiple nodes. Each parser is implemented to parse different type of logs (application logs, file system logs, job logs, and cluster monitoring logs) and load them as a form of Resilient Distributed Datasets (RDDs). The computations for parsing and loading multiple files or separate partitions in each file are distributed and computed in parallel in multiple cores and multiple nodes. Then, parsed results are loaded into memories in multiple nodes or saved in multiple files. By combining the functional programming operators, PATHA provides performance analyses on different types of logs and measurements in scientific cluster.

At the analyzer level, we provide the components of execution time analysis, data dependency performance analysis, and interactive visualization framework as shown in Fig. 2. The framework provides predefined set of functions to enable users to conduct the performance analysis. RDDs loaded as a form of rows of tuples can be computed in parallel by using the functional programming operators such as map, reduce, 'group by key', or 'and sort by key'. In addition, computations between RDDs such as join are supported. Users can interactively conduct performance analysis either querying results or generating graphs by combining with grouping, aggregation, filtering operations with the interesting fields or variables. This is to pinpoint the bottleneck locations in the execution time analysis and identify the most significant field in the data dependency analysis. In addition, it provides the platform that users can use existing libraries of machine learning and statistics in popular programming languages, Java and Python, so that they can easily conduct feature selection, clustering, classification, or regression analysis. Not only conducting our predefined performance analysis, users can implement their customized analysis by combining the libraries on the loaded RDDs without consuming much time on implementation of distributed and parallel programming. The computations at the analyzer level are distributed and computed in parallel in multiple cores and multiple nodes similarly at the parser level. Apache Spark is deployed in a separate cluster with several hundred nodes so that users can interactively execute analyses after connecting to the cluster. [1] The crucial point is that underlying parallel execution of PATHA is dispatched in multiple nodes and multiple cores in each node without the user intervention. Therefore, PATHA can handle large scale performance logs and measurements.

As for the example of the PTF application, the PTF application logs are stored in the database. They are queried with simple conditions such as dates that can reduce the size of the query results. For execution time analysis, timestamps, job id, task id and checkpoint id are loaded into RDDs. The execution time at each checkpoint is computed for each job and task. Then, the execution times are grouped by different keys, e.g., job or task, and the average execution times are computed with the keys. For this grouping, RDDs are needed to include columns with distintive values to be used as keys such as job id, task id and checkpoint id. During the computation for the average, missing timestamps or unordered timestamps are filtered out. These irregularities are caused by various reasons, e.g., failures in the executions at application level or system level. Filtering out these would be challenging and costly to implement using database query or customized user application. For data dependency performance analysis, the execution times are computed with multiple associated

[1] The current version of Apache Spark is optimized for local file system instead of parallel file system in scientific clusters. However, the most performance analysis of PATHA is compute bound since most data movement happens in parsing and loading time.

variables or fields that are potentially related to the identified performance bottlenecks. With the interactive visualization support, PATHA can identify key variables directly related to the performance bottlenecks.

The performance analysis in Sec. IV was conducted with our interactive visualization framework of PATHA, shown in Fig. 8. It provides the visualization tools and figure outputs by allowing users to integrate performance analysis with iPython and web browser. Users can conduct execution time analysis by querying different types of graphs such as histogram, bar graph, box plot and scatter plot. This analysis framework not only allows users to uncover performance bottlenecks in terms of execution time, but also allows them to further query and research possible sources of additional performance bottlenecks related to the data dependency. The development of this tool will continue to advance future research of performance behavior characteristics.

## IV. PTF Case Study

### A. Test Setup

To evaluate PATHA, we used the PTF logs collected on NERSC Edison supercomputer from Mar. 19, 2015 to Jul. 18, 2015 (PST). The PTF application was executed on the compute node with two 12-core CPUs, Intel xeon E5-2695 and 64 GB memory. We also used Apache Spark [27] to distribute and parallelize computational loads for PATHA. It allows more thorough investigation on the PTF application measurements and derived values from the measurements such as the average execution time by averaging differences of the measured timestamps in multiple tasks in each job. PATHA for the experiments was running on a cluster with several hundred machines with two 8-core CPUs, Intel Xeon E5-2670 and 64 GB memory.

### B. PTF Application

Astrophysics is transforming from a data-starved to a data-swamped discipline, fundamentally changing the nature of scientific inquiry and discovery. Currently there are four large-scale photometric and spectroscopic surveys which generate and/or utilize hundreds of gigabytes of data per day. One of them is the Palomar Transient Factory (PTF) which focuses on expanding our knowledge of transient phenomena, such as supernova explosions and massive star eruptions [14]. The transient detection survey component of PTF is performed at the automated Palomar Samuel Oschin 48-inch Schmidt telescope equipped with a camera that covers a sky area of 7.3 square degrees in a single snapshot. Data taken with the camera are transferred to NERSC Edison where a realtime reduction pipeline is run. The pipeline matches images taken at different nights under different observing conditions and performs image subtraction to search for transients. The transient candidates out of this pipeline then pass through machine-learning classifiers to be prioritized for real transients over artifacts. The final output is then displayed through a web portal for visual inspection by human. This pipeline has achieved the goal of identifying optical transients within minutes of images being taken.

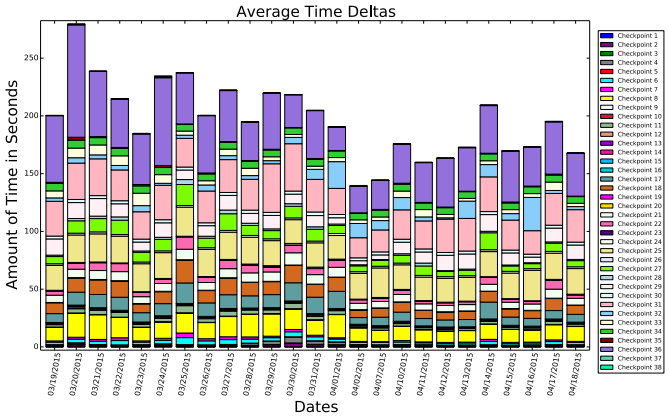



Fig. 3: The average amount of time in seconds that each operation takes. Each color represents one of the 38 checkpoints.

To evaluate PATHA, we used the PTF Application for the experiments. The execution of PTF application involves the executions of multiple jobs. Each job computes different areas, and it consists of 10 tasks whose checkpoints are stored in database when each processing step is conducted. As shown in Fig. 3, the PTF analysis pipeline consists of 38 checkpoints, with each color representing a different checkpoint. Fig. 3 depicts the average amount of time in seconds that the PTF analysis pipeline took on each day to execute all jobs and tasks. From Fig. 3, it is evident that the top five checkpoints with the longest execution time in the PTF pipeline are checkpoints 8, 25, 29, 31, and 36. The average daily percentage calculations taken over a span of 64 days reveal that checkpoint 8 takes on average 7.29%, checkpoint 25 takes 11.16%, checkpoint 29 takes 6.22%, checkpoint 31 takes 14.79%, and most notably, checkpoint 36 takes 23.72% on average. The three checkpoints that took the longest average execution times were further investigated for a potential bottleneck where performance could be improved.

### C. Execution Time Analysis

Next, we dive into the time measurements of checkpoint 36, the Transients in the Local Universe (TILU) query - a geometric query that correlates the incoming candidates with the table of known galaxies with their elliptical shapes and orientations. Fig. 4 shows the box plot of average execution time of this query together with the performance outliers as red dots. We see that many jobs took much longer time than the average. From Fig. 4, we note that certain days, such as March 20, 2015, have larger variance and many outliers. However, the day with the largest number of outliers, March 26, 2015, actually does not have extremely high average execution time, nor even necessarily high variances either. This piqued our interest and we will next examine the execution time on this day more carefully.

Fig. 5 shows a scatter plot of the amount of time in seconds for each job throughout the day, starting at approximately

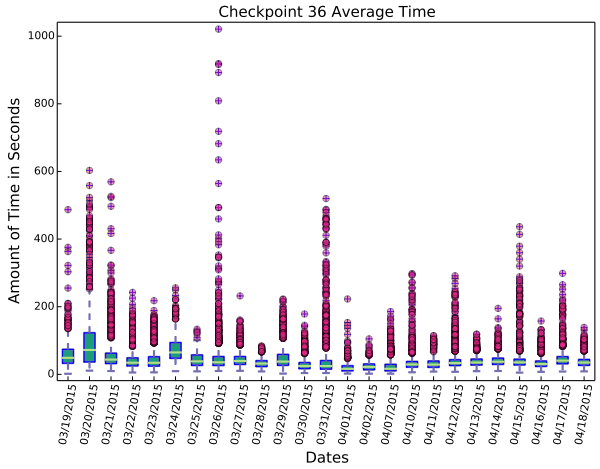



Fig. 4: The amount of time in seconds of each job of checkpoint 36, where each vertical line is for one day, the light green line marks the median time, the blue brackets mark the IQR, the high whisker is at $Q3 + 1.5 \times IQR$, and the red dots mark the instances with extra long execution time.

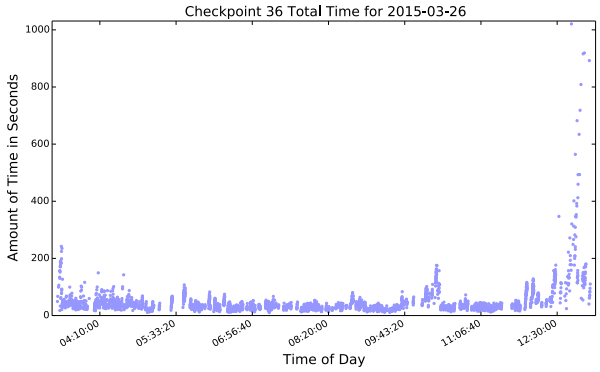



Fig. 5: The amount of time in seconds per day for each job of checkpoint 36, highlighting the average amount of time.

03:00 when the first task of checkpoint 36 was executed on that day. It shows that an execution time spikes during the time period from 12:20 to 13:06 on March 26, 2015. Next, we look more carefully into this time window.

By focusing on the executions in specific time showing significantly more execution times, we can discern whether bottlenecks are caused by cluster load competing system resources or caused by application-specific reasons Fig. 6 shows the time spent by each instance of TILU query in the time window of 12:20 to 13:06. The length of each bar in Fig. 6 reveals the total execution time of each job and its corresponding PTF field. The jobs with longest execution time have job ID 3182, 3189 and 3193 corresponding to PTF fields 3292, 3291 and 14984 respectively. Interestingly, the other PTF fields executed in the similar time window with these PTF fields show much smaller execution times. Since these instances of long execution time are interspersed with very normal looking instances, we speculate that their long execution times are not caused by system loads due to

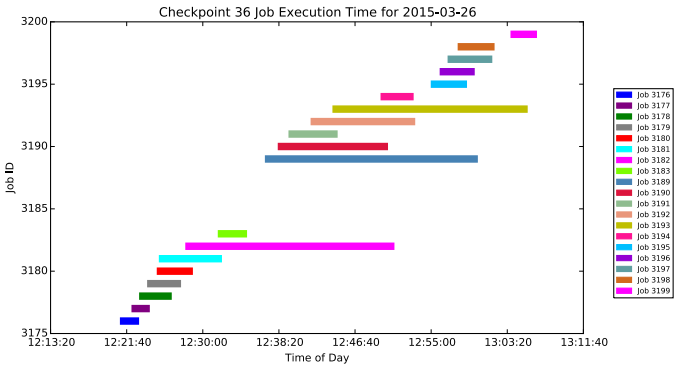



Fig. 6: The execution times of all jobs with their corresponding PTF field during the time period 12:20 to 13:06 on March 26, 2015.

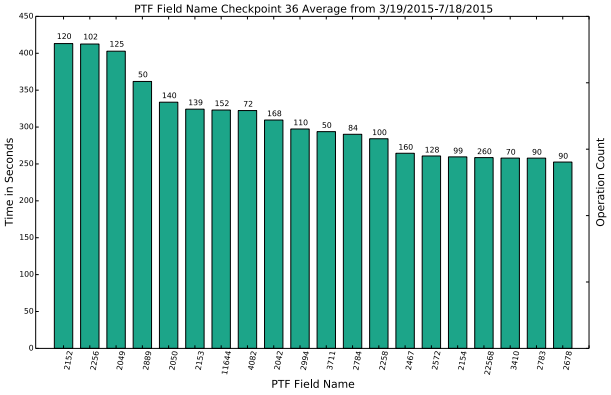



Fig. 7: The average execution times of all PTF fields from March 19, 2015 to July 18, 2015 for checkpoint 36.

competing shared resources. Next, we examine the possibility that these long execution times are caused by differences in user data involved in these queries.

### *D. Data Dependency Performance Analysis*

Based on the suggestions from application scientists, we next examine three PTF attributes to see how they affect the execution time. These three attributes are: PTF field, number of saved objects, and the galactic latitude.

*PTF Field:* Fig. 7 shows the average time of TILU queries plotted against PTF fields, from March 19, 2015 to July 18, 2015. From Fig. 7, it is apparent that PTF field 2049 had the longest average execution time.

To supplement the analysis of checkpoint 36 and to determine if other factors were correlated with the long execution time of checkpoint 36, the two checkpoints (31 and 25) with the 2nd and the 3rd longest execution times were further observed in Fig. 8 and Fig. 9. Checkpoint 25 involves running a program called hotpants that performs image subtraction, and checkpoint 31 consists of running a machine learning classifier, the Random forests [6]. Fig. 8 compares all three checkpoints 36, 31, and 25 for execution times of all jobs on March 26, 2015. It shows that the performance outliers exist in checkpoint 25 and 36, while Fig. 9 displays the PTF fields

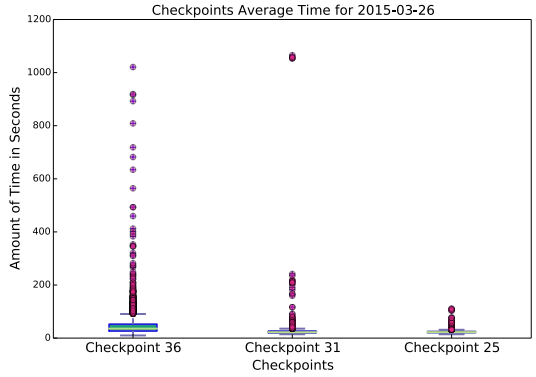



Fig. 8: The execution times of all jobs of checkpoints 36, 31, and 25 on March 26, 2015.

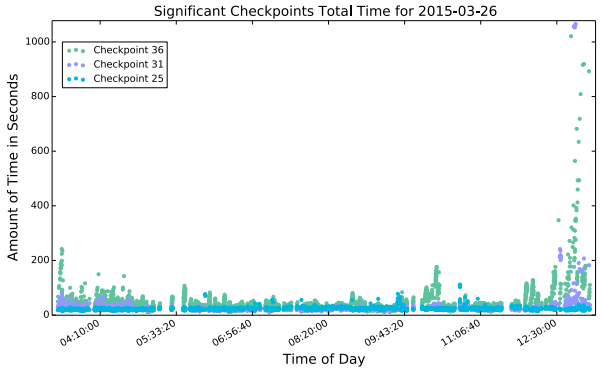



Fig. 9: The execution times of all jobs of checkpoints 36, 31, and 25 during the time 03:00 to 13:00 on March 26, 2015.

shared between all three checkpoints that have the longest average execution time on the same date.

The total execution times of checkpoint 36, 31, and 25 are all shown in Fig. 8 and 9 for March 26, 2015, to allow an easier execution time comparison among the three checkpoints. Using results from Fig. 8 and 9, checkpoints 36, 31, and 25 were further researched by the shared PTF fields, and the PTF fields with the longest execution times on March 26, 2015 are shown in Fig. 8.

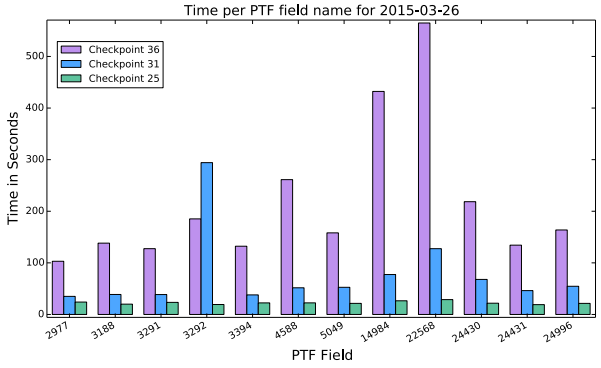



Fig. 10: The average execution times in seconds for PTF fields of checkpoints 36, 31, and 25 on March 26, 2015.

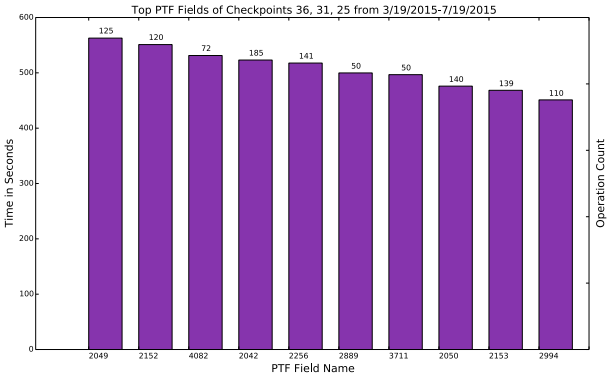



Fig. 11: The top PTF fields that have the longest execution time of checkpoints 36, 31, and 25 from March 19, 2105 to July 18, 2015.

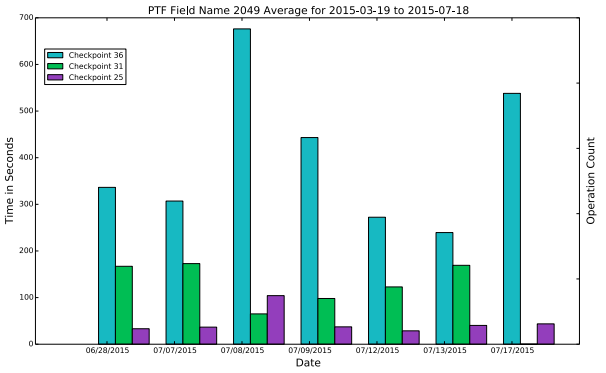
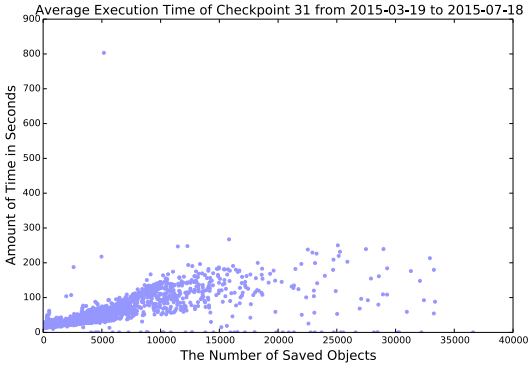


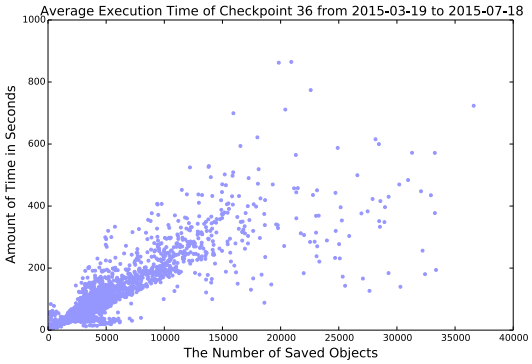

Fig. 12: The average execution times of checkpoints 36, 31, and 25 for each day of PTF field 2049 with the number of measured tasks above each bar.

In addition to the results and calculations shown in Fig. 8 and 9, Fig. 10 was illustrated to find whether any correlations exist between the execution times of the three checkpoints and PTF fields. The executions with PTF field 22568 was shown to have the longest shared average execution time due to the longest execution time in checkpoint 36. Fig. 10 shows that the weak correlation exists, and the execution times of the three checkpoints are not always correlated each other.

In order to further analyze the correlations in these checkpoints, the average execution times associated with the PTF fields from March 19, 2015 to July 18, 2015 were illustrated in Fig. 11 with the number of measured tasks above each bar. This is to compare all three checkpoints with the shared PTF fields that have the longest average execution times. It is important to note that Fig. 11 shows that The executions with PTF field 2049 was calculated to take the longest average execution time of checkpoints 36, 31, and 25 combined. Since the executions with PTF field 22568 showing the longest execution time in Fig. 10 is not shown in this top PTF fields during the 4 months, we further analyzed whether any variances exist in the execution times associated with the PTF fields.

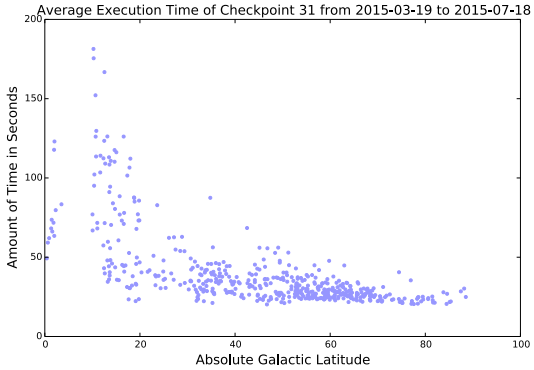



(a) Checkpoint 31

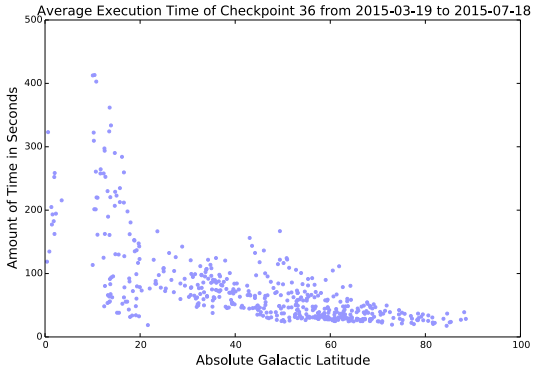



(b) Checkpoint 36

Fig. 13: The average execution time of checkpoints 31 and 36 for each number of saved objects.




(a) Checkpoint 31




(b) Checkpoint 36

Fig. 14: The average execution times of checkpoints 31 and 36 for each absolute galactic latitude.

The executions with PTF field 2049 is illustrated in more details in Fig. 12 with the dates when this field was observed. While the performance bottleneck is related to the PTF fields, the existing variances in the execution times of the particular field confirms that the performance bottleneck is not directly associated with the PTF fields.

*Saved Objects:* In order to find out other directly related variables or fields, we used PATHA to plot more variables with the average execution time depending on the variables. Fig. 13 illustrates the average execution time of checkpoints 31 and 36 for each number of saved objects. While omitted from this Fig., the checkpoint 25 shows the similar pattern as that of 31 and 36. In the PTF application, a fragmentation algorithm is performed on the subtraction images to identify variable stars and transient candidates over the noisy background and to measure their shape parameters such as the length and angle of its major axis and ellipticity. Then, a simple shape cut is applied to remove elongated candidates which are probably artifacts. The candidates that pass the shape cut are saved for further examination, i.e., checkpoints after the checkpoint 25. The reason of different numbers of saved objects is that the total number of candidates for further examination is determined by the number of variable stars (since real transients are rare), which in turn correlates with the total number of stars in a given field. Fig. 13 shows the linear relation between the average execution time and the number of saved objects. [2] It shows the performance bottleneck in these checkpoints when computed with the large number of stored objects. This is because the large number of saved objects requires more comparisons and computation. This identified bottleneck would lead to reduce the computation time when computing with the large number of stored objects.

*Galactic Latitude:* Fig. 14 illustrates a correlation between the execution times of checkpoints 31 and 36 and the absolute galactic latitude (zero degree corresponds to the Milky Way plane). It shows the performance bottlenecks in these checkpoints at low galactic latitudes (checkpoint 25 shows the same performance bottleneck). The physical reason behind it is that the closer a field is to the Milky Way, the more celestial objects, the more transient/variable candidates, and the longer execution time for these checkpoints. At low galactic latitudes, i.e., close to the Milky Way plane, the stellar density is higher, and so is the density of variable stars. Therefore, images taken at low galactic latitudes in general generate more candidates than those at high galactic latitudes.

## V. Conclusion

We developed PATHA (Performance analysis Tool for HPC Applications) using open source big data processing tools. It provides the execution time analysis and data dependency

[2]The linear regression coefficients are $8.515 \times 10^{-4}$ for checkpoint 35 and $5.673 \times 10^{-3}$ for checkpoint 31.

performance analysis on different types of performance measurements from scientific clusters. With the tool, users can identify performance characteristics and performance bottlenecks in their science applications and scientific clusters. As the computations for the analysis are distributed and parallelized in multiple nodes, the framework can handle the measurements from large applications in exa-scale clusters. In a case study involving the PTF application, we identified performance bottlenecks in checkpoints 25, 31, and 36. We also identified their direct data dependencies on the number of saved objects and the absolute galactic latitude. Developers of the PTF application have been working on optimizing identified performance bottlenecks. As the future work, we will extend PATHA to combine the measurements of hardware executions in clusters and the measurements from the applications. In addition, we will automate the process of bottleneck identification. These will help identify the performance bottlenecks due to the system related issues along with the application related issues.

## VI. Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, the U.S. Dept. of Energy, under Contract No. DE-AC02-05CH11231. This work used resources of NERSC.

## References

[1] "Nersc edison," https://www.nersc.gov/users/computational-systems/edison/, 2015.

[2] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCTOOLKIT: tools for performance analysis of optimized parallel programs," *Concurr. Comput. : Pract. Exper.*, vol. 22, no. 6, pp. 685–701, 2010.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. USENIX, Dec. 2004, pp. 259–272.

[4] P. Bod, U. C. Berkeley, M. Goldszmidt, A. Fox, U. C. Berkeley, D. B. Woodard, H. Andersen, P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter," in *EuroSys'10: Proceedings of the 5th European conference on Computer systems*. New York, New York, USA: ACM, Apr. 2010, pp. 111–124.

[5] D. Bohme, M. Geimer, F. Wolf, and L. Arnold, "Identifying the Root Causes of Wait States in Large-Scale Parallel Applications," in *Proceedings of the 2010 39th International Conference on Parallel Processing*. IEEE, 2010, pp. 90–100.

[6] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[7] H. Brunst, M. Winkler, W. E. Nagel, and H.-C. Hoppe, "Performance optimization for large scale computing: The scalable vampir approach," in *Computational Science-ICCS 2001*. Springer, 2001, pp. 751–760.

[8] M. Burtscher, B.-D. Kim, J. Diamond, J. McCalpin, L. Koesterke, and J. Browne, "PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[9] J. Cao, D. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Performance modeling of parallel and distributed computing using pace," in *Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International*, Feb 2000, pp. 485–492.

[10] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control." in *OSDI*, vol. 6. USENIX, 2004, pp. 231–244.

[11] R. Duan, F. Nadeem, J. Wang, Y. Zhang, R. Prodan, and T. Fahringer, "A hybrid intelligent method for performance modeling and prediction of workflow activities in grids," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 339–347.

[12] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft, Oct. 2009.

[13] I. Jolliffe, "Principal Component Analysis," in *Wiley StatsRef: Statistics Reference Online*. John Wiley & Sons, Ltd, 2014.

[14] N. M. Law, S. R. Kulkarni, R. G. Dekany, E. O. Ofek, R. M. Quimby, P. E. Nugent, J. Surace, C. C. Grillmair, J. S. Bloom, M. M. Kasliwal, L. Bildsten, T. Brown, S. B. Cenko, D. Ciardi, E. Croner, S. G. Djorgovski, J. v. Eyken, A. V. Filippenko, D. B. Fox, A. Gal-Yam, D. Hale, N. Hamam, G. Helou, J. Henning, D. A. Howell, J. Jacobsen, R. Laher, S. Mattingly, D. McKenna, A. Pickles, D. Poznanski, G. Rahmer, A. Rau, W. Rosing, M. Shara, R. Smith, D. Starr, M. Sullivan, V. Velur, R. Walters, and J. Zolkower, "The palomar transient factory: System overview, performance, and first results," *Publications of the Astronomical Society of the Pacific*, vol. 121, no. 886, pp. pp. 1395–1408, 2009.

[15] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. B. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[16] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 22:1–22:11.

[17] A. Matsunaga and J. A. B. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 495–504.

[18] F. Rusu, P. Nugent, and K. Wu, "Implementing the palomar transient factory real-time detection pipeline in GLADE: Results and observations," in *Databases in Networked Information Systems*, ser. Lecture Notes in Computer Science, vol. 8381, 2014, pp. 53–66.

[19] S. S. Shende and A. D. Malony, "The TAU parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[20] A. Shoshani and D. Rotem, Eds., *Scientific Data Management: Challenges, Technology, and Deployment*. Chapman & Hall/CRC Press, 2010.

[21] E. Thereska and G. R. Ganger, "Ironmodel: robust performance models in the wild," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 253–264, Jun. 2008.

[22] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter, "The netlogger methodology for high performance distributed systems performance analysis," in *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, Jul 1998, pp. 260–267.

[23] M. Tikir, L. Carrington, E. Strohmaier, and A. Snavely, "A genetic algorithms approach to modeling the performance of memory-bound computations," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, p. 47.

[24] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009.

[25] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP'09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, Oct. 2009, pp. 117–131.

[26] W. Yoo, K. Larson, L. Baugh, S. Kim, and R. H. Campbell, "ADP: automated diagnosis of performance pathologies using hardware events," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE*, vol. 40. New York, New York, USA: ACM, Jun. 2012, pp. 283–294.

[27] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USENIX, 2010.